AA 2019-20

FONDAMENTI DI INFORMATICA

L A B O R A T O R I O

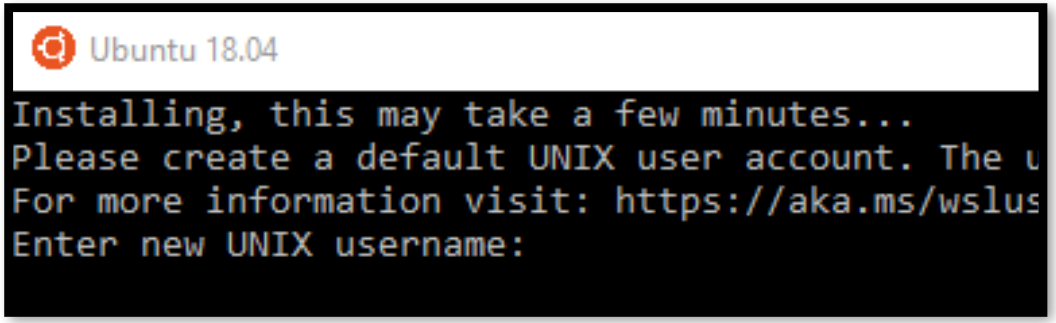
[prof.ssa RAFFAELA MIRANDOLA]

POLITECNICO
MILANO 1863

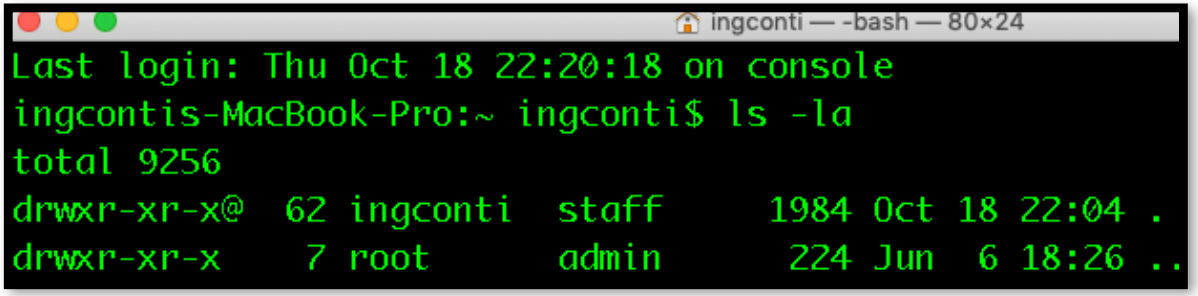
Ricapitolando

- **macchina virtuale**

- **Shell Ubuntu di Microsoft**



- **shell “nativa” OSX-Linux**



Ripasso comandi shell

`ls` `ls -la`	Listare il contenuto di una directory
`mkdir`	Creare una directory
`pwd`	Individuare quale sia la directory attuale
`cd`	Cambiare directory
`rm`	Rimuovere un file/directory

Compilazione

```
gcc -Wall hello.c -o hello
```

- **gcc** → *chiamo il compilatore*
- **-W**(*arning_*)**all** → *attivo la modalità che rende espliciti tutti gli errori (anche quelli potenziali)*
- [*nome_file_sorgente_da_compilare.c*] → *è il file che contiene il codice, quindi è il file con estensione ".c"*
- **-o**(*utput*) [*nome_file_binario*] → *specifico il nome che avrà l'eseguibile*

Controllo dei risultati

```
gcc -Wall hello.c -o hello
```

Se il compilatore non dà errori, allora controllo il contenuto della directory:

```
ls -la
-rwxr-xr-x    1 ingconti  staff     4248 Oct 18 22:04 hello
-rw-r--r--    1 ingconti  staff       13 Oct 18 22:03 hello.c
```

Trovo il codice sorgente e l'eseguibile, come ci si aspetterebbe.
Avvio l'eseguibile:

```
./hello
```

Errori comuni

Principali errori riscontrati durante la prima sessione, in ordine crescente di pericolosità

1 – Rientro del testo (o *indentazione*)

Di per sé non un “vero” errore, ma impedisce la lettura ed è sintomo di:

- approccio poco sistematico al problema

- scarsa padronanza delle logiche del flusso del programma

1 – Rientro del testo (o *indentazione*)

don't

```
int main()
{int a=0;


if(a==0) {
a=b;
}


…

if(a>0) {
     ...
}
     else {
          ...
     }
```

do

```
int main()
{
     int a=0;
     …

     if(a==0)
     {
        a=b;
     }

     if(a>0)
     {
        …
     }
     else
     {
        ...
     }
}
```

do (*alternativa*)

```
int main(){
     int a=0;
     …

     if(a==0){
        a=b;
     }

     if(a>0){
        …
     }
     else{
        ...
     }
}
```

2 – Errori logici di costrutto

don't

```
if(a>0)
{
      …
}
else (a<=0)
{
      …
}
```

La negazione di **a>0** è già minore o uguale a 0.
Difatti è inutile anche:

```
if(a>0)
{
     …
}
else
{
     if (a<0)
     {
        …
     }
      else
     {
        if (a==0)
```

do

```
if(a>0)
{
      …
}
else
{
      …
}
```

```
if(a>0)
{
     …
}
else
{
     if (a<0)
     {
        …
     }
      else
     {
        …
```

3 – Errori logici di assegnazione

don't

```
int a=1, b=2;
…..
if (a = 0)
{
}

..
if (b = a)
{
}

..

int trovato = 0;
while(trovato = 0)
{
}
```

Quanto vale qui *b* alla fine?
ATTENZIONE: "=" assegna, "==" confronta.

do

```
int a=1,b=2;
…..
if (a == 0)
{
}

..
if (a == b)
{
}

..

int trovato = 0;
while(trovato == 0)
{
}
```

4 – Sviste logiche sulla negazione

don't

```c
int main() {

    int a = 100;
    int b = 1;

    if (a=!b){
        printf("hello\n");
    }

    printf("a is %d\n", a);
    return 0;
}
```

output:

```
a is 0
```

do:

```c
int main() {

    int a = 100;
    int b = 1;

    if (a!=b){
        printf("hello\n");
    }

    printf("a is %d\n", a);
    return 0;
}
```

output:

```
hello
a is 100
```

5 – Errori sui cicli

don't

```
while(x>0);
{
}
```

```
for (x=1; x<100; x++);
{
}
```

Il ";" crea un blocco di statement del ciclo vuoto

do

```
while(x>0)
{
}
```

```
for (x=1; x<100; x++)
{
}
```

6 – Errori compilazione e shell

a) **`gcc hello`**

b) **`./hello.c`**

c) Far partire **`./hello`** senza aver ricompilato

a) **`gcc hello.c`**
(*produce* **`a.out`**)
Ancora meglio: **`gcc hello.c -Wall -o hello`**

b) **`./hello`**

c) In due passi:
`gcc hello.c`
`./hello`

Esercizio 0: Numeri primi

Scrivere un programma che, dato un numero intero positivo inserito dall'utente, stampi a video "primo"se tale numeri è primo.

Esempio:

```
Inserisci: 3  -> "primo"
Inserisci: 4  -> "non è primo"
Inserisci: 37 -> "è primo"
```

Suggerimenti:

- un numero è primo se è divisibile solo per sè stesso e per 1
- usare l'operatore `%` per calcolare il resto della divisione intera, fra il numero e i precedenti.

Esercizio 1: Array

Si scriva un programma che legga da input due array A di 10 elementi e B di 5 elementi.
Il programma stampi: “CONTIENE”
se A contiene la sequenza contigua dei numeri di B.
ES:
A= [3,4,66,77,88,9,33,11, 66,100]
B=[77,88,9,33,11]
output: “CONTIENE"
ma non stampa nulla per B = [77,88,9,34,11]

Esercizio 2: Matrici

Scrivere un programma in linguaggio C che legga in input dei numeri **float** e riempia una matrice 4x4.
(usare #define)

il programma deve stampare tutti gli elementi delle celle della diagonale principale, ossia con indice riga == indice colonna.

es 1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

output:

1
6
11
16

Esercizio 3: Matrici

Scrivere un programma che usi due matrici NxN, dette A e AT.

L' utente inserisce i valori di A e il programma riempie AT in modo

che sia la Trasposta di A (e la stampi).

https://it.wikipedia.org/wiki/Matrice_trasposta

$$A = \begin{pmatrix} 2 & 4 & 8 \\ 3 & 2 & 0 \\ 5 & 3 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad A^T = \begin{pmatrix} 2 & 3 & 5 & 0 \\ 4 & 2 & 3 & 1 \\ 8 & 0 & 1 & 0 \end{pmatrix}$$

Esercizio 4: Matrici

Scrivere un programma in linguaggio C che legga in input due matrici 4x4, dette A e B

il programma deve stampare tutti gli indici riga/colonna che dove

le celle di A hanno lo stesso valore delle celle di B nelle posizioni corrispondenti

Mat. A

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Mat. B

1	9	10	4
55	0	7	4
19	22	22	12
3	4	5	88

output:

(0,0)
(0,3)
(1,2)
(2,3)

Esercizio 5: Matrici

Scrivere un programma in linguaggio C che legga in input una matrice A quadrata N x N

il programma deve cercare tutti i massimi locali (ossia i valori che sono il massimo delle celle adiacenti) e sostituirli con zero.

(oss: ne caso di riga/colonna 0, si consideri la posizione N-1, sia dall' alto lato)

A

1	**9**	**10**	4
55	-5	7	4
19	**23**	22	12
3	4	5	**88**

output

1	**0**	**0**	4
0	-5	7	4
19	**0**	22	12
3	4	5	**0**